Managing Embedded System Software

with the

RFI-ES Development Tools

020310
DCV

Table of Contents

Managing Embedded System Software
with the
RFI-ES Development Tools

1.0 Background - The WindRiver Tools

The WindRiver product called Tornado is an integrated development environment for VxWorks. Tornado uses the gnu compiler, linker and other tools to convert C and C++ source code into object modules that can be loaded into the embedded system's memory with the aid of the VxWorks target resident dynamic linking loader. The gnu tools are configured as cross compilers for a specific target microprocessor when they are built for the host where they are run. Further configuration of the gnu tools is provided at runtime through shell environment variables and compiler switches that must be set up by the user. The environment variables configure the compiler to emit code suited for a specific microprocessor and version of VxWorks. Fortunately the Fermilab personnel who maintain the WindRiver tools have created a series of scripts for initializing the environment variables for the microprocessors used at Fermilab. These scripts are made available to users through a set of aliases that are exported into each user's environment at login time. The aliases can be viewed by typing the following shell command:

```
alias | grep env
```

The compiler switches cause the compiler to properly parse the language being compiled and configure the linker to create the proper object module format. Unfortunately the user must configure the compiler switch settings in a file called Makefile that is processed by the UN*X make utility. With this simple scheme a Makefile is written for each processor or target to be supported.

The process for creating code for any given target is to: 1) write a Makefile for the target, 2) invoke the proper env_xxx alias to set up the proper gnu tools for the target and 3) invoke make to run the compiler and linker against the project source files. This process must be repeated for each target to be supported.

The RFI department's embedded system development tools (hereafter referred to as 'the tools') assist the developer by combining the Makefiles for one or more targets into a single file and by automating the process of building for multiple targets.

2.0 Projects

The tools define a model where software is developed in manageable modules referred to as projects. Projects are characterized through the use of two configuration files: **Targets** and **Makefile**. Both configuration files are required and must reside alongside the project's source code within the project's repository. See section 3.3 for a description of the Targets file format and section 4.0 for a description of the Makefile file format.

3.0 Using the Embedded System Development Tools

The tools are located and managed in the directory **/home/rfies** on the development computer **nova.fnal.gov**. The software repository is managed by CVS (or optionally SCCS) and its location is transparent to users. The default configuration of the tools supports the UN*X **bdrfinst** group. Members of other UN*X groups may tailor the tools as described in section 3.1 below.

The tools should be used with the **bash** shell – the use of other shells is not recommended because of the potential for diminished functionality. Users can switch to the bash shell temporarily by simply calling it from the command line. If bash is not your default shell, making it so requires the assistance of the development computer's System Administrator. Bash requires two files for initialization: **.bash_profile** and **.bashrc**. A third file **.bash_aliases**[1] may also be required. Examples of these files can be copied from the **/home/rfies** directory. Edit your copy as necessary to tailor the shell to your tastes.

The following several paragraphs describe the procedure for obtaining and tailoring the tools for individual preferences. If developers are willing to use the default configuration (all user's embedded system development files in ~/esd and subdirectories thereof) they can source the script **/home/rfies/esd/useresdconfig** from the bash shell to create the default directory structure and make copies of all required files. A related note "Configuring Accounts on Nova" describes the procedure for creating the default user configuration. Users following the procedure outlined in the note may skip ahead to the discussion of the setup tool in section 3.2 below.

[1] .bash_aliases is not actually required by bash but is called by the example .bashrc file and so it is included here.

3.1 Tailoring the Tool Environment

To gain access to the tools and repository users must copy the file **/home/rfies/esd/examples/useresdsetup.bash** to their own scripts directory, optionally edit the copy to establish their personal characteristics, and execute the updated copy with the shell's source command. There are four definitions in the example useresdsetup.bash file that may be modified by the user:

• **USER_SCRIPT_DIR** must point to a directory that contains 'callback scripts' that the tools can call when tailoring tool performance to individual user's needs. By default USER_SCRIPT_DIR is set to ~/esd/scripts.

• **USER_SANDBOX_DIR** must point to a directory that contains one subdirectory for each of the user's projects. The convention is to have a subdirectory for each project with the same name as the project (i.e., project foo would be in a directory called $USER_SANDBOX_DIR/foo.) By default USER_SANDBOX_DIR is set to ~/esd/src.

• **USER_ORGANIZATION** must contain the name of the organization that the user is affiliated with. This variable is used to identify where projects are located within the source code repository's directory tree. By default USER_ORGANIZATION is set to rfies.

• **USER_REPOSITORY** must contain one of: CVS, SCCS or SCM. This variable tells the tools which source repository system the user prefers, allowing the tools to create aliases that simplify the use of the chosen repository. By default USER_REPOSITORY is set to CVS.

Figure 1 represents an example useresdsetup.bash script containing definitions for a user that is using the default definitions. Since these values are provided by the tools as defaults their specification by this user is redundant. By the way, the name useresdsetup.bash is not sacred, users may rename their copy as desired since they are the only individuals referencing this file.

```
#!/usr/local/bin/bash
#
#
# Filename: %M% - %Q%
# Revision: %I%
# Date and Time: %G% %U%
# Id : $Id$
#
#
# Description:
#   Script to set up the rfies group's embedded system
#   development tools on behalf of the user.
#   This script should be sourced by the caller.
```

```
#

#
# set up the users' development organization
#
export USER_ORGANIZATION=rfies

#
# set up where tools find:
#      project_xxx scripts
#
export USER_SCRIPT_DIR=~/esd/scripts

#
# set up where tools find:
#      directories, with same name as projects, which contain project source
#
export USER_SANDBOX_DIR=~/esd/src

#
# set up users repository preference (e.g., CVS, SCCS or SCM)
#
export USER_REPOSITORY=CVS

#
# set up embedded system development tools
#
source $(ESD_BASE_DIR)/scripts/esdsetup.bash

#
# End of script
#
```

Figure 1

Once useresdsetup.bash has been sourced the full set of tools and the software repository will be available to the user.

3.2 Setup Tool – use: setup projectName [targetName] [headerType]

The setup tool allows users to easily initialize the development environment and switch to the sandbox directory for project projectName. Setup **MUST** be used to move from project to project since it establishes the proper values for environment variables used in the project build process. The optional targetName parameter allows for specifying the target to be used in subsequent make operations. If targetName is not specified the target specified in the Target file (last target built) will be selected. If the Target file is missing the default target (the first target listed in the Targets file) will be selected. The optional headerType parameter allows for specifying the headers directory to be used in

subsequent make operations. If headerType is not specified the headers specified in the Headers file (last headers used) will be selected. If the Headers file is missing the production headers will be selected. NOTE: If the headerType parameter is used then the targetName parameter must also be specified.

If the setup tool does not find projectName in the user's sandbox it will check the repository. If the repository contains projectName setup will offer to make a working copy of that project in the user's sandbox. If projectName is not found in the repository setup will offer to create a new sandbox for the user. If the user chooses to create a new sandbox setup will offer to install one of: VME, Slot-0 or DSP oriented versions of template Makefile and Target files.

The setup command does the following for the user:

1 - provides the value of environment variable PROJECT
2 - provides a value for environment variable USER_PROJECT_DIR
3 - provides a value for environment variable TARGET
4 - sources $USER_SCRIPT_DIR/project_projectName iff it exists
5 - creates handy aliases for using the repository with the project
6 - runs the target tool
7 - runs the headers tool
8 - creates gprojectName alias for quick transfers to the project directory
9 - does a cd to the project sandbox

In step #4 the script $USER_SCRIPT_DIR/project_ projectName may be used to specify special values for USER_PROJECT_DIR or TARGET, or to define project specific environment variables. Setting USER_PROJECT_DIR will allow the source code for the project to reside in a location other than the customary $USER_SANDBOX_DIR/$PROJECT location. Setting TARGET will specify a default target for builds other than the first target listed in the Targets file. An example project_projectName file may be copied from **$(ESD_BASE_DIR)/examples/project_xxx**. Figure 2 represents an example project_projectName script for a project with the source in the standard location and to be built for the PPC603 target by default.

```
#!/usr/local/bin/bash
#
#
# Filename: %M% - %Q%
# Revision: %I%
# Date and Time: %G% %U%
# Id : $Id$
#
```

```
#
# Description:
#   Set up environment for project  xxx.
#   Sourced by the ESD_SCRIPT_DIR\setup.bash script.
#

#
# users may specify the source directory for $PROJECT
#
USER_PROJECT_DIR=$USER_SANDBOX_DIR/$PROJECT

#
# users may specify the target which they want to build by default
#
TARGET="PPC603"

#
# users may specify any project specific shell variables
#

#
# End of script
#
```

Figure 2

3.3 Target Tool – use: target targetName

The target tool allows users to specify a new target for subsequent make activities. The tool searches the Targets file (described below) for a target specification matching targetName and if a match is found initializes the tool environment for that target. Tool target also places the specified target name in the Target file. Since the Target file is in the dependency list for all object modules this will guarantee that the project will be rebuilt if the target changes.

The Targets file has the following format:

```
# Targets for project
targetName  setupScriptName  [flag...]
```

Each line of the Targets file not beginning with a # represents a target definition containing one or more fields. The targetName field is required and gives a name to the target being defined. Target names are all capitalized by convention and generally reflect some significant characteristic of the target such as its CPU or SBC (e.g., PPC603 or MVME2301.) All other fields are optional. The setupScriptName field contains the file name specification for the setup script that configures the tools for the specific processor and version of VxWorks required by the specified target (e.g.,

/usr/local/bin/wind2_PPC603.bash.) The various forms of this file name specification can be determined by typing the following shell command:

```
alias | grep env
```

The flag field represents zero or more flags of the form xxx=yyy which will be passed to the Makefile on the make command line as the target is being built.

The target tool is intended for use when interactively building a project with make.

3.4 Headers Tool - use: headers production|test|development

The headers tool places the specified header type in the Headers file. Since the Headers file is in the dependency list for all object modules this will guarantee that the project will be rebuilt if the header type changes.

The headers tool is intended for use when interactively building a project with make.

3.5 DSP Install Tool - use: dinst production|test|development ldrName.ext

Because some DSP projects do not follow the convention of having a sandbox for each project the setup and make tools cannot install DSP loader files. To provide automated install facilities for DSP loader files the setup and make tools are replaced with a single tool called **dinst** (Dsp INSTall.) The dinst tool implements production, test and development make rules for DSPs. The setup and make tools are of no use when working on DSP projects that are not located in conventional sandbox directories.

The first parameter to dinst is the install directive:

- **production** – Install the DSP loader file in the production DSP library and any header files in the production header library. Also create symbolic links from the production libraries to the test libraries. This rule effectively promotes the project from test to production.
- **test** – Install the DSP loader file in the test DSP library and any header files in the test header library.
- **development** - Install the DSP loader file in the development DSP library and any header files in the development header library.

The second parameter, ldrName.ext, is the name of the loader to be installed. For example the command:

```
dinst production  mixfr.ldr
```

Will install mixfr.ldr in the appropriate production DSP library.

Since the dinst tool uses the UN*X make utility each DSP project requires a Makefile. An example Makefile may be obtained from **$(ESD_BASE_DIR)/examples/dsp/Makefile**. The dinst tool uses the target tool to establish the target DSP so each DSP project also requires a Targets file. An example Targets may be obtained from **$(ESD_BASE_DIR)/examples/dsp/Targets**.

3.6 Help Tool - use: helpme

Since additional tools are likely to be added to the system over time the helpme tool prints short (hopefully) helpful hints about the various tool commands and their parameters.

4.0 Building Software Modules with make

Converting collections of C and C++ source files into useful software modules is accomplished with the aid of the standard UN*X make utility. UN*X make provides a mechanism for stating the rules by which software modules are built and then automatically applying those rules to invoke the language compilers and linkers in the proper sequence to produce loadable object modules. Many software modules contain code that can be used in more than one application or executed on more than one processor. Such general modules can be rebuilt independently for each ‘target’ with the aid of make.

WindRiver provides a set of make include files that simplify the creation of user Makefiles but unfortunately they only support builds of single targets. The functionality of the WindRiver make include files has been extended by the build.mk make include file to provide rules for compiling mixed C and C++ sources and for building object modules for any specified (and known to the tools) target. The build.mk make include file supports high-level specification of the composition of a software module thereby allowing Makefiles to be quite uncomplicated in their appearance.

Figure 3 contains an example Makefile for a project that supports more than one processor and target. The example appears to be quite lengthy because it includes examples of definitions for multiple processors and targets. Careful inspection reveals that most of the definitions in the example are null and could be removed.

```
#
#
# Filename: %M% - %Q%
# Revision: %I%
```

```
# Date and Time: %G% %U%
# Id : $Id$
#
#
# Description:
#   Makefile for projects.
#

# specify  sources which must be compiled
C++SOURCES = $(wildcard *.cpp)
CSOURCES =  $(wildcard *.c)
# specify  all header files to be installed in the includes directory
HEADERS = $(wildcard [!_]*.h)

# specify  all startup  script files to be installed in front-end
# download  directory
SCRIPTS = $(wildcard *startup)

# specify  compiler parameters  which affect  all builds
LIBRARIES =
INCLUDES =
DEFINES =
CFLAGS =
C++FLAGS = $(CFLAGS)

# specify additional compiler parameters which affect specific processors
CPU_xxx_LIBRARIES =
CPU_xxx_INCLUDES =
CPU_xxx_DEFINES =
CPU_xxx_CFLAGS =
CPU_xxx_C++FLAGS = $(CPU_xxx_CFLAGS)

# specify additional compiler parameters which affect specific targets
TARGET_xxx_LIBRARIES =
TARGET_xxx_INCLUDES =
TARGET_xxx_DEFINES =
TARGET_xxx_CFLAGS =
TARGET_xxx_C++FLAGS = $(TARGET_xxx_CFLAGS)

# use character '@' for quiet makes, leave blank to detail make process
OUT = @

#
#  End of user portion of makefile -- include xxx.mk include files below
#
include build.mk    # rules for building projects
include install.mk  # rules for installing  projects into libraries

#
# End of makefile
#
```

Figure 3

An example Makefile may be copied from **$(ESD_BASE_DIR)/examples/Makefile.**

The make tool provides the following definitions on the compiler command line so that your source code can determine the environment for which it is being compiled:

- -DCPU=xxxx,
- -DOS_VERSION=yyyy, and
- -DTARGET=zzzz

where for example xxxx could be MC68020 or PPC603, yyyy could be VW_531 or VW_54 and zzzz could be MC68040 or VXICPU030. The full set of CPU and OS_VERSION definitions is dependent upon the WindRiver tools. The full set of known TARGET definitions can be determined by inspecting the file targets.h in the project rfiessupport.

4.1 Make Switches

The make system supports optional switches for modifying the normal make process.

The project switch tells the make system which project is being built. The value provided will override the PROJECT environment value. This switch is intended for non-interactive use by other tools.

The target switch tells the make system which target is being built. The value provided will override the TARGET environment value. This switch is intended for non-interactive use by other tools.

The headers switch tells the make system which header file library to include in the include file search path:

- **headers=production** – use the production header library.
- **headers=test** – use the test header library.
- **headers=development** – use the development header library.

If these switches are unused the default action is to use the production header library.

The DSP object identification switch explicitly tells make which object file to install in cases where there may be more than one object file per project:

- **file=fileSpec[.ext]** – install the specified object file.

If this switch is not used the default action is to install projectName.ldr. If the extension to the fileSpec operand is not specified .ldr will be assumed. This switch applies only to DSP object module install operations.

4.2 Make Rules

The make system provides a set of rules for building projects and another for installing projects.

The build rules, provided by build.mk, direct the compiler and linker to produce object modules that support the specified CPU, OS version and target. The build rules include:

- **make –** Build (i.e., make all) the project for the currently specified target.
- **make clean** – Remove all generated files (i.e., .o, .a, .doc, and munching files) from the cwd.
- **make doc** – Make document file for all source files in the project.
- **make echo** – Print a list of the project's header, source and script files.
- **make help** – Print help information about the make rules.
- **make info** – Print information about the current project configuration.
- **make librarydirectory** – Create production, test and development library directories on fecode-bd for the current target. If the directories already exist nothing will be altered. This is useful when building for a previously undefined target.
- **make lint** – Run the lint program on all project C and C++ sources.
- **make map** – Produce linker map file.
- **make <file>.cppsym** – List all preprocessor symbols for the specified source.
- **make <file.ext>.doc** – Make document file from the specified file.
- **make <file>.lint** – Run the lint program on the specified source.
- **make <file>.out** – Compile and munch single source file into <file>.out.
- **make <file>.pp** – Produce preprocessor output only for the specified source.
- **make <file>.s** – Produce assembly source only for the specified source.

The install rules, provided by install.mk and dspinstall.mk, use shell commands to copy the project's header and object files into the appropriate libraries. The install rules include:

- **make downloaddirectory** – Create a download directory on fecode-bd for the current project. If the directory already exists nothing will be altered. This is useful when setting up a new front-end for downloading.
- **make librarydirectory** – Create production, test and development library directories on fecode-bd for the current target. If the directories already exist nothing will be altered. This is useful when building for a previously undefined target.
- **make install -** Print help information about the make install rules.

• **make installscript** – Install all specified script files into the project's download directory.
• **make production** – Install the project's object file in the production library and its header files in the production header library. Also create symbolic links from the production libraries to the test libraries. In the case of projects that are to be installed into download directories, the object module is placed in the download directory and a symbolic link is made from libxxx.out to testxxx.out. This rule effectively promotes the project from test to production.
• **make test** – Install the project's object file in the test library and its header files in the test header library. In the case of projects that are to be installed into download directories, the object module is placed in the download as testxxx.out rather than libxxx.out.
• **make development** - Install the project's object file in the development library and its header files in the development header library. In the case of projects that are to be installed into download directories, the object module is placed in the download as devxxx.out rather than libxxx.out.

5.0 Environment Variables

The tools reference and define several environment variables. The variable name, location defined and short description of each variable follows.

USER_ORGANIZATION
Initialized in useresdsetup.bash.
Provides the name of the organization for which the tools are configured (e.g., rfies or pbares). This organization is used by the tools to properly locate include and library directories, and to search the repository for projects related to that organization.

USER_SCRIPT_DIR
Initialized in useresdsetup.bash.
Points to location where the tools can find (optional) user supplied tool callback scripts.

USER_SANDBOX_DIR
Initialized in useresdsetup.bash.
Points to location where the tools can find sandbox directories for the user's working set of projects. By convention each directory in the sandbox has the same name as the project that it contains.

USER_REPOSITORY

Initialized in useresdsetup.bash.
Indicates which repository the user wishes to use (e.g., CVS, SCCS or SCM.)

USER_PROJECT_DIR
Initialized in setup.bash – optionally tailored to user requirements in user's project_projectName callback script.
Points to the sandbox directory containing the user's working copy of the current project.

ESD_BASE_DIR
Initialized in esdsetup.bash.
Points to the base directory for tool operations. This directory contains subdirectories containing all tools, scripts and the SCM code repository.

ESD_SCRIPT_DIR
Initialized in esdsetup.bash.
Points to the directory containing tool scripts.

ESD_DOWNLOAD_DIR
Initialized in esdsetup.bash.
Points to the base directory of the project download directories. Each front-end project has a download directory containing the operating system, startup script and project object module to be downloaded at boot time.

ESD_DSP_DIR
ESD_TESTDSP_DIR
ESD_DEVDSP_DIR
Initialized in esdsetup.bash.
Points to the base directory of the DSP loader libraries.

ESD_LIBDSP_DIR
Initialized in esdsetup.bash.
Points to the base directory of the DSP shared libraries.

ESD_LIB_DIR
ESD_TESTLIB_DIR
ESD_DEVLIB_DIR
Initialized in esdsetup.bash.
Points to the base directory of the project object libraries.

ESD_INC_DIR
ESD_TESTINC_DIR
ESD_DEVINC_DIR
Initialized in esdsetup.bash.
Points to the directory containing the project public header files.

PROJECT
Initialized in setup.bash – optionally tailored to user requirements in user's project_projectName callback script.
Contains the name of the current project.

TARGET
Initialized in setup.bash – optionally tailored to user requirements in user's project_projectName callback script.
Contains the default target name for the current project.

End.